The state of computing in the humanities: making a synthesizer sound like an oboe

Sperberg-McQueen, C. M.

Veröffentlichungsversion / Published Version
Zeitschriftenartikel / journal article

eine zusätzliche etatisierte Personalstelle bewilligt, um die Fortführung dieser Arbeiten zu sichern. Diese Förderung schuf u.a. die Voraussetzung für die (und verpflichtete gleichzeitig zur) Weitergabe dieser Erfahrungen und Werkzeuge in größerem Stil auch an andere Hochschulen und Forschungseinrichtungen.

Inzwischen bestehen Sammellizenz-Vereinbarungen für TUSTEP mit etwa 100 Hochschulen, Akademien und anderen Forschungseinrichtungen im In- und Ausland. An einigen Universitäten haben sich ähnliche zentrale Anlaufstellen wie in Tübingen gebildet, die sich auf dieses Werkzeug stützen. Vor allem auswärtige TUSTEP-Nutzer haben sich vor zwei Jahren in der »International TUSTEP User Group« ITUG zusammengeschlossen, die sich laut Satzung insbesondere um »Förderung der Ausbildung von TUSTEP-Anwendern, ... die Fortbildung von erfahrenen Benutzern sowie ... die Förderung des Informationsaustauschs zwischen den TUSTEP-Anwendern« kümmert.

In der Abteilung LDDV sind wir uns bewußt, daß das, was wir selbst dazu beitragen konnten, einschließlich der Leistungen des hier entwickelten Werkzeugs, der guten und selbstverständlichen, teilweise recht kontinuierlichen Zusammenarbeit mit vielen Projekten über Instituts- und Fakultätsgrenzen hinweg (und über Tübingen hinaus) zu verdanken ist, wie sie in den Geisteswissenschaften nicht gerade sprichwörtlich ist.

Dieses Kolloquium, an dem einige von Ihnen seit dessen Einrichtung heute vor 22 Jahren teilnehmen, ist ein aktueller Beweis dafür. Dafür mochte ich den Teilnehmenden auch im Namen meiner Mitarbeiter herzlich danken.

The State of Computing in the Humanities: Making a Synthesizer Sound like an Oboe

C. M. Sperberg-McQueen (University of Illinois, Chicago)*

1. Introduction

In 1970, when the Division of Literary and Documentary Data Processing (the anniversary of which we are celebrating here today) was organized here in Tübingen, humanities computing differed from what we see today, but it was not exacdy in its infancy. If we date the inception of the field from 1948, when Father Roberto Busa persuaded Thomas J. Watson, Sr., of IBM to support Busa's work on the *Index Thomisticus,* then we can reckon twenty-five years

*** Protokoll des 65. Kolloquiums über die Anwendung der EDV in den Geisteswissenschaften an der Universität Tübingen am 18. November 1995.**

from today back to the founding of the Division, and another twenty-three years from the founding of the Division, back to the beginning of the field. By this reckoning, the Division is now two years older than was the field itself, on the day the Division was organized. Not every institutional arrangement in this field has achieved this venerable antiquity, and congratulations are in order.

It is a useful exercise to stop at anniversaries like this one and consider where we are, where we have come from, and where we are going.

In this talk, I'd like to review briefly some of the most salient accomplishments in literary and linguistic computing. From them, it seems to me we can learn some things about the field, and about the texts which are the objects, in part, of its study. In the light of these considerations, we should be able to see better where we need to go next, and where there are opportunities for useful new accomplishment.

2. Looking Back

The field of humanities computing, as I define it, began with the efforts of Father Roberto Busa to create an exhaustive word-list to the complete works of St. Thomas Aquinas — that is, with the production of a concordance. That concordance took thirty years, and the last volume of the *Index Thomisticus* appeared in 1978, in what a form much evolved from the original plan, created with tools (such as high-level programming languages and computer-driven photocomposition machines) which, Busa reminds us in his retrospective essay on the project, had not yet been invented when the project itself began.

In the 1950s, the production of concordances remained the most visible activity in this field.[1] John W. Ellison concorded the Revised Standard Version of the Bible; Busa performed pilot studies on four hymns written by Aquinas and on the Dead Sea scrolls. And the first volume of the Cornell Concordance series appeared, in which concordances for important English and American authors poured forth in abundance throughout the following decades. Stylistic and stylometric studies, word frequency counts and similar efforts were also prominent.

Equally important for my theme, though, is the revolution in linguistics introduced by the publication of a slim volume by Noam Chomsky called *Syntactic Structures.* Chomsky champions the use of formal methods for linguistic study, and his words are worth quoting:[2]

---

[1] **My chronological account is indebted, among others, to D. M. Burton's articles on "Automated Concordances and Word Indexes: The Fifties" and "Automated Concordances and Word Indexes: The Early Sixties and the Early Centers," *Computers and the Humanities* 15 (1981): 1-14 and 83-100.**

[2] **Noam Chomsky, *Syntactic Structures,* Janua Linguarum Series Minor Nr. 4 (The Hague: Mouton, 1957), p. [5].**

> Precisely constructed models for linguistic structure can play an Important role, both negative and positive, in the process of discovery itself. By pushing a precise but inadequate formulation to an unacceptable conclusion, we can often expose the exact source of this inadequacy and consequently, gain a deeper understanding of the linguistic data. More positively, a formalized theory may automatically provide solutions to many problems other than those for which it was explicitly designed.

In the 1960s, scholars in the field founded many research centers (some of which, like the Division in Tubingen, survive, many of which do not) and journals such as *Computers and the Humanities.* Larger collections of electronic text were created, some as repositories for independently produced texts (such as Project Libri at Dartmouth), some as homogeneous collections (such as the Trésor de la langue française). Concordances were produced for many authors. Standards were created for character sets; conventions for modern European languages, however, were still poor. In 1967, the Cocoa (Word COunt and Concordance program on Atlas) program was developed at Atlas Computing Laboratory in England. And language corpora were developed, including the Brown Corpus of American English, and the Pfeffer corpus of German.

I need hardly describe the developments of the more recent period in detail; Prof. Ott has just done so. Since 1970, the emphasis on concordances and other tools for textual study has been joined by other activities, but never wholly pushed aside. And our software tools have become stronger, over time.

What have we accomplished in these fifty years?

First, we have made great efforts to produce concordances, word indices, and similar tools. A vast number of such reference works have been produced — so many that they appear to have no news value anymore, and some skeptics discount them entirely as research products, most recently and visibly Mark Olsen, my neighbor in Chicago.[3]

In a way, of course, he is right. Early concordances were newsworthy, but the reference materials we produce today have no news value. Why? Because they are commonplace. But that is an accomplishment, not a shortcoming. Concordances have become routine: there are humanists now without any interest in computing per se, who routinely begin their work on a new text by scanning a good modern edition (for personal use only) and generating a full-text concordance to the work, to aid the study of style, theme, characterization, and plot. That was not routine even twenty years ago. Transforming the concordance from a work of piety accorded only to the most sacred texts into a working tool to be applied, in passing, to any text we propose to study seriously — that is a major achievement.

---

[3] **In his essay "Signs, Symbols, and Discourses: A New Direction for Computer-Aided Literature Studies," *Computers and the Humanities* 27 (1993): 309-14.**

Another major achievement are the procedures for automatic and semi-automatic lemmatization, developed initially for concordances which lemmatize and sometimes annotate each word form, but now used primarily perhaps in language corpora. The Brown Corpus, the Lancaster-Oslo-Bergen Corpus, and the British National Corpus (to name just the most prominent English-language corpora) all provide lemmata and morphological and word-class analysis for their texts.

More purely literary algorithms have also not been lacking: the method used by Joseph Raben and Philip Smith to find echoes of one author in another, the methods developed by John B. Smith and others to study thematic patterns in literary works, and the extensive work on the quantitative description of style which still dominates the pages of our journals, are all fundamentally literary or textual, rather than linguistic, in their goals. Textual software for concordance programs, for collation of textual witnesses, and for scansion and metrical analysis, study of rhyme technique, etc., in different languages, are all literary in their uses.

Literary and linguistic computing has also benefited from software developments outside the field. Database software can be used for literary and linguistic as well as commercial data. The relational model of databases shows the advantages to be gained from applying formal principles to practical problems. Text processors have an obvious importance to a discipline which studies texts, though some aspects of current word processors are a mixed blessing at best. The older generation of stand-alone text formatters, which applied the traditional notion of markup to electronic manuscripts, laid the groundwork for a theory of descriptive markup, which I regard as a definite advance. And our field can also exploit software for network delivery of information, first widely demonstrated by Gopher in the early 1990s, and more recently by the World Wide Web.

In humanities computing, as elsewhere, software can benefit from careful engineering. In some, but not all, humanities computing software we can see outstanding examples of standard software engineering ideas:

- a consistent tool set
- consistent interfaces (both to the user and to other programs)
- support for handling the character set problems of computing today
- support for serious intellectual work
- and a consistent data format, ensuring that the output of any program can be read by any other program.

Concordances and similar reference works are a major accomplishment of our field, but the software may be more important, because with software we can create many reference works.

More important than the software, however, are the electronic texts themselves.

It took time for this idea to spread, and it may not be universal even now. Jess Bessinger and Philip Smith almost threw away their electronic text of

Beowulf because the concordance was finished and they saw no other use for it. We are painfully similar to the humanist printers who used priceless Carolingian manuscripts to set type, then threw them away because they saw no further use for them. We can guess, but cannot know, what posterity will require from the work we do today. And perhaps our guesses about posterity's needs are only recapitulations of our own desires.

Still, it has become clear that our electronic texts last longer than the reference works we create from them, and much longer than our software, which must be changed or rewritten constantly for new operating environments. The *Index Thomisticus,* the Thesaurus Linguae Graecae, the Trésor de la langue française, and the many texts in the Oxford Text Archive are all tools for which future scholars will have reason to be grateful. And while the others are completed, the Oxford archive continues to grow, at last count to 1500 texts, in all about one and a half gigabytes of data.

Electronic dictionaries are another resource to cherish. The electronic Oxford English Dictionary was constructed by the publisher, not by humanists. But the electronic forms of other dictionaries — Lexer's dictionary of Middle High German, Liddell and Scott's dictionary of Greek, and many other dictionaries of many languages, are owed only to our colleagues in our own field.

These resources have changed the nature of research in some disciplines. The Thesaurus Linguae Graecae has revolutionized textual exegesis in classics: lexicology, the study of realia, thematic investigations, all have a much sturdier foundation than before the TLG. It can also transform the teaching of classics: texts from the TLG form the heart of the Perseus Project, in which they are linked to translations, commentary, search engines, morphological analysis software, glossaries, and extensive collections of maps and images of vases and other physical objects. Collections like Perseus could dramatically improve school and university education in all languages and literatures.

Linguistics has undergone its own transformation, with the large-scale development corpora, from the Brown Corpus of Modern American English of the 1960s, to its contemporary successors, some of them 100 times the size of their forebear.

The persistence of corpus builders in the face of skepticism has, I am told, changed the face of large portions of linguistics. The ease with which corpus linguists have been able to destroy utterly the hypotheses of linguists who relied solely on native-speaker intuition, by adducing counter-examples or by showing the non-occurrence of theoretically predicted forms, has effectively persuaded many practitioners that in serious linguistic work, linguistic intuition by itself is just not up to the job. Better results are achieved when a linguist begins with intuition, but then examines the relevant examples from a corpus to check intuition against the data. The data seldom fail to improve on intuition alone.

Such a revolution is not quite so visible in literary and other textual studies, but I think that the gradual profusion of concordances and electronic texts has

placed limpets on the hull of impressionistic scholarship which will ultimately take that ship completely out of the water. This will be cause for rejoicing, I think, among those who believe, as I was taught by Helmut Tervooren, a medievalist in Bonn (confronted with an imaginative but implausible interpretation of Hartmann von Aue):

Phantasie ist gut. Philologie ist besser.

3. Approaching the Oboe

But useful as reference works, software, and electronic texts all are, they don't fully explain the attraction of literary and linguistic computing. Why do some of us care so much about it?

As a graduate student, I once met a musicologist named John Chowning. He was interested in electronic music, and when I asked him what he was working on, he said they were trying to make an electronic synthesizer sound like the various instruments in an orchestra or voices in a choir a trombone, an oboe, a soprano singing *ah, eh, ih, oh, oo.* That sounded interesting, I said insincerely, but wasn't it rather an expensive way to go about making an oboe sound? Wouldn't it be easier and cheaper to get an oboist and get the sound live? He gave two reasons to do it electronically, and both apply to our field.

First of all, no, it is not usually cheaper to hire an oboist Oboists are expensive. So are clarinets, French horns, bassoonists, string players, and percussionists. If synthesizers could simulate these sounds, many composers could hear their work who will never hear it otherwise. Student composers who write symphonies, or even octets, cannot afford to hear them performed by live musicians. With the synthesizer, they can hear their work.

But there is another reason for trying to make a synthesizer sound like an oboe, and this is what I think of as Chowning's Principle.* We know when we start that an oboe sounds the way it does because of the particular form taken by the sound waves it produces. To replicate the sound, we record it identify the characteristic wave form, analyse it using Fourier analysis, and use the synthesizer to create a similar wave form. The only problem is that the result may sound only distantly like an oboe. In isolating the characteristic wave form of the oboe, we may have simplified it to remove inessential random variations in the sound, which (we now discover) were not inessential after all. When we restore them and replicate the more complicated wave form, it sounds a bit more like an oboe.

But it sounds, perhaps, only like the *middle* of a note: the beginning and the end do not sound at all like an oboe. In this way we discover that the turbulent confused wave forms we see before the wave has achieved its "characteristic

*** N.B. The description of sound analysis given here is not misleading as far as I know, but it should not be taken as a literal description of the sounds of the oboe.**

form" are themselves characteristic of the oboe. They, too, must be modeled, giving a better simulation of an oboe, but still an imperfect one. We must study yet again the actual oboe sound, and how it differs from our imitation. This cycle of simulation, failure, and renewed study may never end. We can go on indefinitely studying the sound of the oboe and trying to make our synthesizer sound just like it.

Our work, therefore, yields not just a simulated oboe sound, but a deeper understanding of the sound. If we had decided that there was no point in replicating the oboe mechanically, or that doing so was inappropriate, then we would still "know" that replicating the characteristic wave form was all there was to it The attempt to demonstrate our understanding by replicating the sound has shown us just how complex the natural phenomenon really is, and how little we actually did understand at the outset. As Roberto Busa said here five years ago:[5] "In fact analyzing texts leads to realizing the presence of the mystery of God at the roots of human understanding and speaking."

Several points might be made about this story and the idea of making a synthesizer sound like an oboe.

First, technology need not deepen the gap between the poor and the rich. The synthesizer makes it easier, not harder, for students to hear their work. The World Wide Web makes it easier, not harder, to publish specialized work with limited readership and no commercial prospects.

Second, we learn most from a firm, even a dogmatic, insistence on perfecting the simulation. Consider an example from the history of my own field. Modern Germanic philology began with the careers of Karl Lachmann and the Brothers Grimm. Among other things the Grimms identified the systematic sound shifts which mark the Germanic language family, which we know today as Grimm's Law.

The neo-grammarians *(Junggrammatiker)* of the late 19th century devoted a vast amount of effort to isolating further sound laws, and to proving that sound laws have no exceptions. Modern linguists sometimes mock this principle as too rigid, but they seem to me to miss an essential point. It was the insistence on this principle that made it troubling to the neo-grammarians to observe that many sound laws seemed to have a number of exceptions after all. Worst of all, Grimm's Law is among them. And it was because the neo-grammarians were so troubled that the Danish philologist Karl Verner had this problem running through his head, the day he noticed that the exceptions to Grimm's Law all shared a common phonological feature in Indo-European. There are no "exceptions" to Grimm's Law after all: the apparent exceptions are simply instances of Verner's Law.

---

[5] **"Half a Century of Literary Computing: Towards a 'New' Philology,", in "Reports from Colloquia at Tübingen," *Literary and Linguistic Computing* 5 (1990): 69-73, here p. 70.**

Verner's Law would never have been discovered had the neo-grammarians not insisted on the completeness of their theory. It is the push to make the synthesizer sound not just a little bit like an oboe, but *like* an oboe, that teaches us about the sound of oboes; it is the insistence on eliminating all exceptions that led to the discovery of Verner's Law.

In literary and linguistic computing, our synthesizer is a computer. Our oboe is the text we study. How well are we doing in making our synthesizer sound like an oboe?

The short answer is: we're making progress, but we still have work to do. As I describe our progress, please bear in mind that if I insist on attempting to capture *all* of a text — its every aspect, its every nuance, this is not due to a naive assumption that it is simple, or even possible. Texts are not simple. Representing texts in computers in appropriate forms is not simple. But by ignoring, from time to time, such pragmatic questions, we can improve our understanding.

So what does our oboe sound like?

4. Document Geometries

Implicit in the software of the last decades one can identify several different models of text, of increasing subtlety. Major textual resources of this period exhibit less apparent progress, not because they still use primitive models of text, but because many major projects insisted from the beginning on better models, even if that meant writing software from scratch for each project. The models of text implicit in textual resources may indeed have declined, as scholars grew less willing to write software for themselves and were forced to accept the simplistic models of general-purpose software.

Each of these models of text corresponds to a particular idea of the structural characteristics of texts, which can be expressed as a sort of grammar for texts.

4.1. Rectangular Text

Many early programs regard text as *rectangular.* Vertically, the text is a sequence of lines (which looks a lot like a deck of punched cards), horizontally each line is a sequence of characters. Lines are often atomic: they may be replaced as units, but not modified internally.

Formally, the grammar for this model is fairly simple:

text ::= record+

record ::= CHARACTER+

That is, a *text* is one or more occurrences of *record. A record* is one or more occurrences of *CHARACTER. CHARACTER* is undefined -i.e. we accept the notion of character as a primitive, basic concept This is conveyed here both by the absence of a formal rule for *character* and by the convention of writing it in upper case.

4.2. Linear Text

Later programs, especially interactive editors, chose to regard text as *linear*. In this model, text is a stream of characters. There is no notion of lines or records, carriage-return or new line characters are simply characters in the stream, like any others. This model is generally regarded as a step forward, since it seems to correspond better with the nature of text. In some fundamental way, that is, text does appear linear. Formal grammars, for example, treat sentences as strings — i.e. linear streams — of words. And as a rule, we have no hesitation in specifying, for any word or character of a text, what the *next* word or character is.

Formally, this model of text is even simpler than the preceding one: text is one or more characters, in arbitrary sequence.

text ::= CHARACTER+

This formal simplification can also be used to simplify the user interface: without the added layer of the line, it is no longer necessary to provide two distinct sets of commands for adding, deleting, or modifying lines and for performing the same operations on individual characters within lines.

The linear model is found in a great deal of software still in use today, including the well known editor emacs and (with appropriate elaborations, described below) the word processors on most desktops today.

4.3. Vertical Text

Linear texts are frequently enhanced with morphological or other annotation. The text proper is regarded as a linear stream of words, and the annotation is arrayed alongside it as a second stream beside the first. Most users of this extended linear model make the text run vertically, one word per line, with the surface form in one column and the analysis in a second (or third, ...). The result is a rectangular text of a new sort, which we may call a "vertical" text format.

Vertical formats have been in use at least since the 1970s. The tagged version of the Brown Corpus of Modern American English has two columns, word on the left and part-of-speech tag on the right. The University of Pennsylvania's Computer-Assisted Tools for Septuagint Studies (CATSS) materials also use vertical formats. An elaborate vertical format is used by the Lancaster-Oslo-Bergen corpus of British English.[6] The interlinear text annotation software developed at the Summer Institute of Linguistics uses a similar two-dimensional model (with horizontal, not vertical, text).

---

[6] **Stig Johansson, in collaboration with Eric Atwell, Roger Garside, and Geoffrey Leech, *The Tagged LOB Corpus* (Bergen: Norwegian Computing Centre for the Humanities, 1986). The corpus is available in both vertical and horizontal versions.**

The vertical text makes text linear in one dimension, while exploiting the second dimension to record information not explicit in the surface forms of the text Father Busa refers to this transverse axis as the »internal hypertext of each word.

The individual lines of a vertical text resemble the records of a database, with a record for each word of the text and columns for as many aspects of the word as interest us. General-purpose database tools can work with data in vertical form, though this appears not to be commoa

Formally, the vertical model of text resembles the rectangular model described above, but replaces the *record* with the more complex notion of *word,* which has columns for the surface form and one or more attributes such as part of speech:

text ::= word+
word ::= form attribute+
form ::= CHARACTER+
attribute ::= part of speech code | lemma |... (etc.)
part of speech code ::= NOUN \ VERB \...
lemma ::= CHARACTER+ ...

4.4. Variations on Linear and Vertical Text

4.4.1 Cylindrical and Toroidal Text

The linear and vertical models of text have been extended in a number of ways which ought to be mentioned. It has been suggested[7] that conventional full-screen editors present not a linear text but a rectangular text block in which the user can move up or down, right or left, until one encounters the border of the text In some editors, the user can scroll past the top or bottom of the text; the software "wraps" to the other end of the text again. Text in such editors is not linear or rectangular but circular, or cylindrical. An editor could wrap horizontally, too; its text model would be toroidal, or doughnut-shaped.

This proposal has a light-hearted feeling, but poems can be found in which the text does wrap: the *next* line after the *last* line is the first line of the poem. The toroid (or at least, the cylinder) is a natural representation of the basic form of these texts.[8]

---

[7] **In an oral presentation by Donald Ross at the Modem Language Association in the mid 1980s. As far as I know, he has not published this proposal in written form.**

[8] **In the case of E. E. Cummings' poem "crazy jay blue)", for example, a parenthesis is opened in the last line and ends in the first line. See his *95 poems* (New York: Harcourt, Brace, & World, 1958), poem 5. It is hard to find natural texts which wrap horizontally as well as vertically, but it would require a bold historian indeed to aver that no examples are to be found anywhere among the Western peoples. The closest example I have encountered is the poem "Die Schönheit ist ein Blitz" in Philipp von**

4.4.2 Multiple Witnesses

A number of variants on the linear text have been proposed to handle multiple versions of the same text; these range from the notion of a rectangular text composed of several parallel linear streams, to my own suggestion of text as a sort of Rhine Delta, with several strands or streams, which flow together when all witnesses have the same readings, and separate when there are variants, to reunite when the witnesses once again have the same reading.

4.4.3 Linear Text with Variables

Probably the most widespread adaptation of linear text is visible in Cocoa-style tags.[9] Cocoa-tagged text, like vertical text, supplies one or more streams of information alongside the surface of the text. The vertical text, however, segments all parallel streams into words; Cocoa tagging allows the parallel streams to contain units larger or smaller than words.

In practice, Cocoa tags can be thought of as defining a set of variables, which change values at different points in the text. Cocoa tags do nothing but supply values for these variables. For example, the following lines assign the values "III", "iv", and "Polonius" to the variables *A* (act number), *S* (scene number), and *C* (character):

<A III>
<S iv>
<C Polonius>

Since values may be assigned at any point, to know all the values for a given location in the text we must process the entire text left to right from the start. In contrast, the simpler models of text allow random access to the text, because all the information applicable to any point in the text is explicitly present at that point itself.

Formally, tagged linear text distinguishes *content* (the surface form of the text) from *markup* (tags) and allows them to mix arbitrarily. Cocoa tags, for example, can be described by the following simple grammar:

Zesen's *Poetischer Rosen=Wälder=Vorschmack* (Hamburg: In Verlegung Tobias Bundermans, Gedruckt durch Heinrich Werner, 1642), p. 36. In this sonnet, the second half-line of each Alexandrine line is aligned vertically, and the poem is so constructed that it can be read either horizontally (line by line) in fourteen Alexandrine lines or vertically (column by column, i.e. all the first half-lines, and then all the second half-lines) in fourteen couplets with short lines. I am indebted to M. R. Sperberg-McQueen for acquainting me with this poem.

[9] **Cocoa-style tags are used in Cocoa, the Oxford Concordance Program, and Tact. Similar markup can be used in Ttistep; as a non-expert in the use of Tüstep, I am relying here on Winfried Bader's *Lernbuch TUSTEP: Einführung in das Tübinger System von Textverarbeitungsprogrammen* (Tübingen: Niemeyer, 1995). On descriptive markup in Tüstep, see in particular section 3.5, "Sachlich orientierte Textauszeichnung."**

text ::= (CHARACTER | tag)+
tag ::= '<' variable name ' ' value '>'
variable name ::= CHARACTER
value .:= CHARACTER+

Common batch-oriented formatters modify the Cocoa model slightly: their tags have direct effects on the processing of the text, and may take actions rather than setting variables. Formally, text is for them a long sequence of characters, interspersed with commands:

text ::= (CHARACTER | COMMAND)+

The linear model captures the basic linearity of text; the tagged linear model adds the ability to model, within limits, some non-linear aspects of the text. But it misses another critical characteristic of text Text has structure, and textual structures can contain other textual structures, which can contain still other structures within themselves. Since as readers we use textual structures to organize text and reduce its apparent complexity, it is a real loss if software is incapable of recognizing structural elements like chapters, sections, and paragraphs and insists instead on presenting text as an undifferentiated mass.

4.5. Untyped Hierarchical Text

A hierarchical model of text attempts to represent explicitly the formal structures of text e.g. its chapter, section, subsection, paragraph and sentence units, in order to enable software to exploit that structure for display and processing. To that end, it models text as a sort of tree with nodes and subnodes.

The hierarchical model appears in the NL/Augment system developed in the 1960s by Douglas Engelbart and his associates. In Augment documents are represented as lists of items, which themselves may be either strings of characters, or themselves composed of subordinate lists of items. The individual items might represent chapters of a book, sections of a chapter, paragraphs of a section, or sentences of a paragraph. Because the items of the logical structure are (apparently) not labeled as to their type, we can refer to Augment as using *untyped* hierarchies.

A slightly less pure example of this model can be found in Tustep, with its native hierarchy of page, line, and subline.[10] If the Tustep hierarchy is used to model the formal structure of the text then processing can be specified in terms of the formal units of the text in ways which would be somewhat harder without the logical hierarchy.

---

[10] I call Tustep an impure example because the different levels of the hierarchy have distinct types and because the hierarchy has a fixed depth. It is nevertheless grouped here because the names *page, line,* and *subline* are conventional, not necessarily restricted to actual pages and lines of a text. They can be, and often are, used to model chapters, paragraphs, stanzas, etc.

Both systems also capture some non-linear and non-hierarchical aspects of text NL/Augment provided an early form of hypertext linking, while Tustep provides a form of descriptive tagging already mentioned above.

Formally, an untyped hierarchy can be represented with the following recursive grammar.

text ::= items+
item ::= item+ | CHARACTER+

4.6. Hypertext

Yet another model of text may be found in software designed to support hypertext Defined, in the words of Theodor Holm Nelson, as "non-sequential writing", hypertext requires that the linear sequence normally expressed by adjacency of words or sentences be augmented by other methods of connecting locations; two locations marked as connected in this way are said to be *linked.*[11]

In simple hypertext models, the text as a whole is organized into atomic units (sometimes called *chunks)* connected by links. The typology of links is limited only by the imagination of the user and the patience or skill of the implementors. Links may start from any location within a chunk, but they point only to whole chunks, not to parts of chunks. For this reason, this simple hypertext model, exemplified by the Macintosh HyperCard system, is sometimes called *chunky hypertext.*

In more complex forms of hypertext links may start and end at arbitrary points or spans of text within the system. Chunks play no special role. (To distinguish it from chunky hypertext, this more complex and capable model is sometimes called *creamy hypertext,* borrowing an antonym of *chunky* from the varieties of American peanut butter.)

Hypertext captures the notion of text as a set of units with relations which include the linear sequence and parent/child relations familiar from the other models already described. Formally, hypertext can be represented as a directed graph of nodes linked to other nodes. Unfortunately from a computational point of view, there is no formal way of describing restrictions on the links between nodes, and no formal grammar seems to offer itself as a description of the hypertext model.[12]

---

[11] **Citations seem superfluous, but for a reasonably full account of Nelson's ideas on data structures for text, see his *Literary Machines,* ver. 90.1 (Sausalito: Mindset Press, 1990), published at various times by a variety of publishers in different locations.**

[12] **The structure of individual nodes in a hypertext system may of course be described with a formal grammar, it is the directed graph itself which does not seem to correspond to a grammar in any useful way.**

4.7. Typed Hierarchical Text

Typed hierarchical text resembles untyped hierarchical text but assigns a *type* to each node of the hierarchy. Like tagged linear text it allows characteristics of the text to be marked explicitly, for one word or many. It differs from tagged linear text primarily in organizing the text into a hierarchy of nesting segments, instead of modeling text as an arbitrary mixture of content and tags.

Brian Reid used this model in his text formatting program Scribe. Because Scribe describes text in terms like *section, subsection, paragraph,* or *bibliography,* it is easier for many users than other batch formatters of its day. Perhaps better known today is Leslie Lamport's LaTeX, which takes the same approach, as do the GML (Generalized Markup Language) system developed by IBM and re-implemented at Waterloo.

In each of these systems, a node of the text hierarchy is assigned a type, e.g. *document, front matter, body, back matter, chapter, section, paragraph).* Each type may have a different kind of content. A document, for example, may contain front matter, body, and back matter, in that order, while the body may contain paragraphs and chapters.

In early systems of this type, however, descriptions of the allowable content of each node type were merely informal. There was no automatic checking to see that each type of node actually had the prescribed content. LaTeX, for example, checks to ensure that subsections occur only within sections, and subsubsections only within subsections. But it does not check (or so I am told) to ensure that a new chapter is not begun within a footnote.

We can, however, use standard grammatical formalisms to define the system of node types formally. The resulting document grammar can be used to parse the document and control processing. Document grammars allow software to verify mechanically that the document is formally correct. Some errors must elude such checking, but whole classes of typographic errors can be found without manual inspection.[13]

The best known systems with document grammars are all applications of the Standard Generalized Markup Language, or SGML. In this category belong both the Hypertext Markup Language (HTML) used in the World Wide Web and the markup scheme of the Text Encoding Initiative (TEI), a much more elaborate attempt to define document grammars for use in humanistic scholarship.

---

[13] **The introduction of formal document grammars thus occupies a place in the progress of document processing analogous to the introduction of formal grammars in the development of programming languages. Even if there were no other differences between them, SGML would be an advance over Cocoa for this reason, in the same way that Algol 60 was an advance over Cobol or Fortran. Unlike grammars for programming languages, of course, document grammars may have to describe pre-existing materials.**

The TEI attempts to provide an *explicit* and formal description of the real nature of text. The TEI's formal description is found in the TEI *Guidelines,* and runs to some 1300 pages. For some (though perhaps not all) modern scholars, text is:

- an object with a hierarchically defined logical structure[14]
- and also with one or more physical structures (volume, page, etc.), also hierarchically defined
- a linguistic object
- a rhetorical object
- a formal (literary) object
- with hypertext qualities
- with textual variation.

The entire TEI Guidelines can be summed up in one phrase, which we can imagine directed at producers of commercial text processing software: "Text is not simple."

The TEI attempts to make text complex — or, more positively, the TEI enables the electronic representation of text to capture more complexity than is otherwise possible.

The TEI makes a few claims of a less vague nature, too.

- Many levels of text, many types of analysis or interpretation, may coexist in scholarship, and thus must be able to coexist in markup.
- Text varies with its type or genre; for major types the TEI provides distinct *base tag sets.*
- Text varies with the reader, the use to which it is put, the application software which must process it; the TEI provides a variety of *additional tag sets* for these.
- Text is linear, but not completely.
- Text is not always in English. It is appalling how many software developers forget this.

None of these claims will surprise any humanist, but some of them may come as a shock to many software developers.

5. Looking Forward

Each model in our sequence captures a bit more of the nature of text; we have made some little progress in making our synthesizer sound like an oboe. What remains to be done?

First I think we should use our existing foundations. We need good software, with a good model of text to create concordances and word counts, do interactive search and retrieval, and prepare scholarly editions. We can build

---

[14] **In fact, of course, the logical structure of text cannot always be arranged in a strict hierarchy of nesting objects, and the TEI Guidelines therefore also provide ways of representing non-nesting structures.**

that software from existing software for computing in the humanities, or from existing SGML software. But we must bring SGML awareness and support for humanities computing tasks together in the same package.

We should improve our software, in speed, facilities, and usability.

We should further refine our model of text. We should formalize the semantics of TEI markup, as far as we can. And we should extend the TEI to areas not now covered.

We should build new, larger, and better electronic resources. We should begin by systematically extending our collections of electronic texts. The history of humanity can be read from the texts humans have produced; let us have *all* of them in electronic form. Like the Thesaurus Linguae Graecae, let us begin with the texts most frequently read — but continue with the goal of ultimate completeness.

We should create electronic dictionaries, both new dictionaries and electronic versions of the older scholarly dictionaries we still use. We have the OED in electronic form. We have Lexer's Middle High German dictionary. Let us also create an electronic Grimm. Let us create electronic forms of all of the great dictionaries of all of our languages.

We should build tools for scholars: interactive concordance programs, programs to study and display manuscript readings, perform morphological and lexical and syntactic analyses. With all the respect and admiration I have for Tustep and its creator, I would like to take away the monopoly it currently seems to enjoy as the only text processing program currently available for scholars which deals systematically with the possibility of textual variation.[15]

And most of all, we should be ready to fail, and to fail again. We must be prepared to repeat ad infinitum the cycle of modeling, failure, renewed study, and modeling, in our tenacious attempts to make a synthesizer sound like an oboe, or to make a computer — a physical object with complex electronic circuitry — represent a text, an abstract object with incomparably more complex internal structure and external linkages. That cycle is, perhaps, the technological version of the traditional hermeneutic circle. It is worth while, I think, to wander in circles in this way, seeking a way to make our synthesizers sound like violins, and French horns, and oboes. Because, after all, we all know that when we hear the sound of a solitary oboe playing the note A, the real music of the orchestra concert is about to begin.

---

[15] **Other collation programs exist, but concordance programs and other processing programs do not, with the exception of Tustep, handle textual variants at all.**